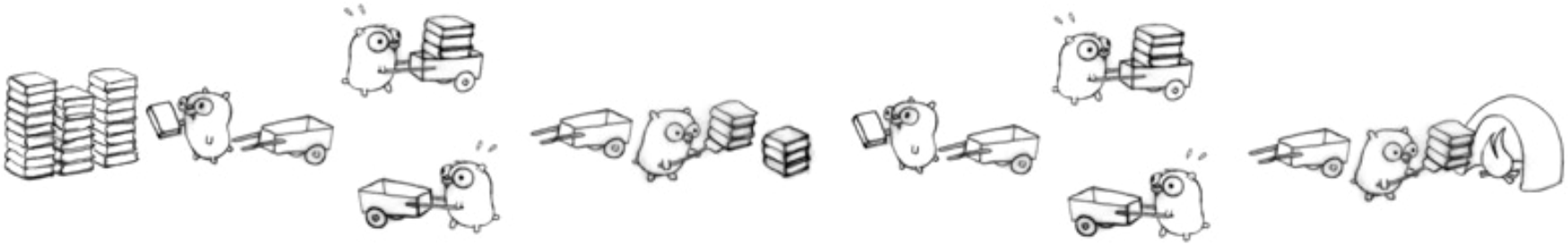
Programmieren mit Go

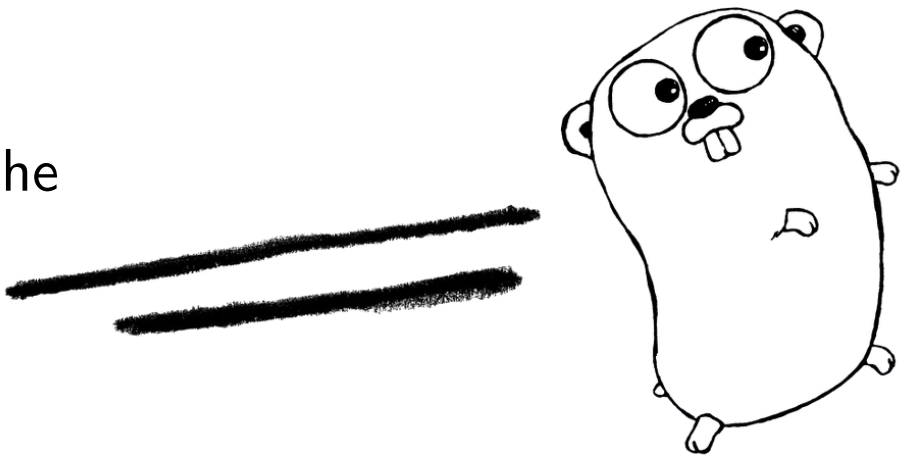
Sebastian ‘tokkee’ Harl
<sh@tokkee.org>

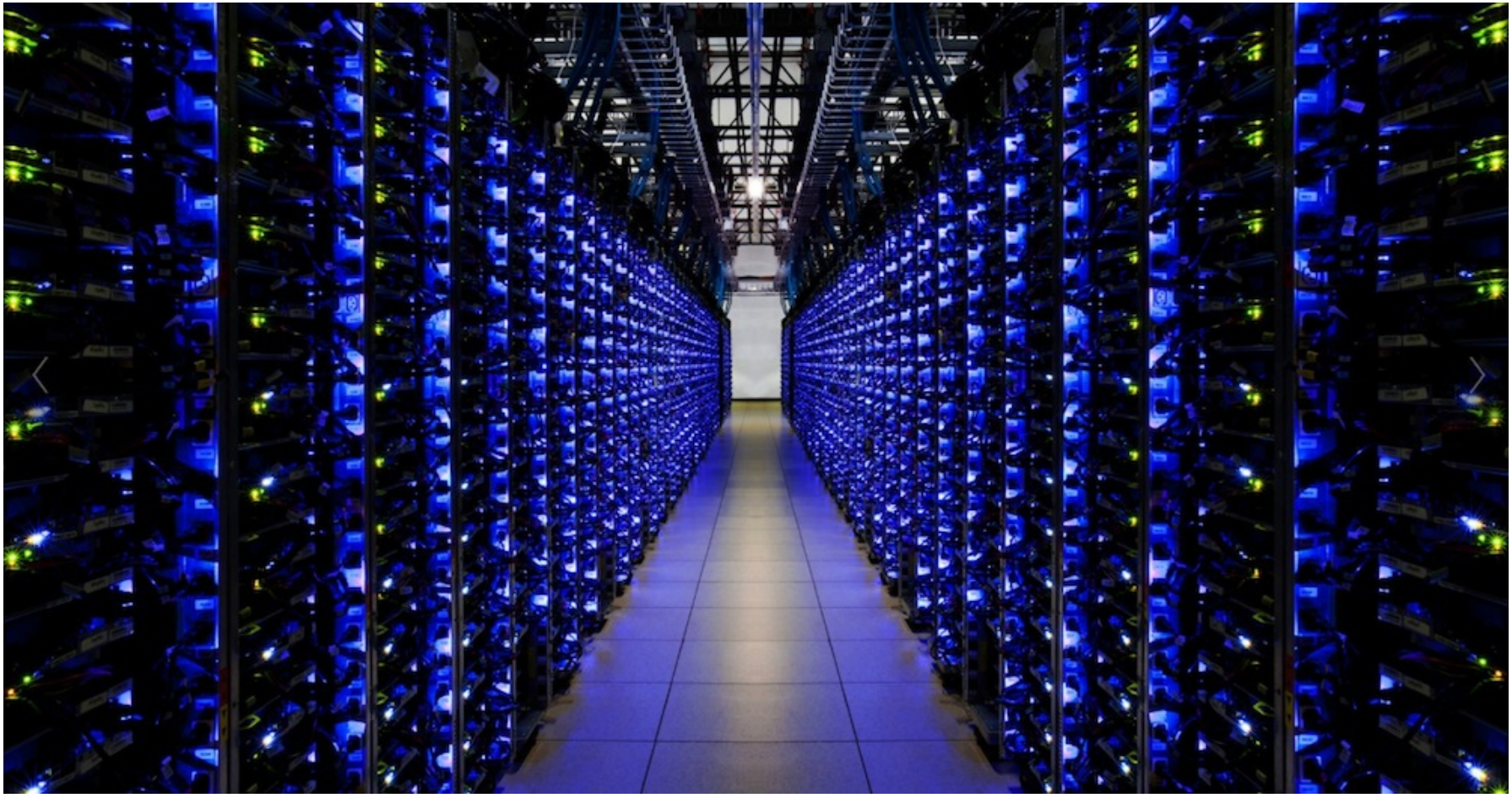


Überblick

Was ist Go?

- `https://golang.org/`
- OpenSource Programmiersprache
- imperativ, Interfaces, Pakete
- statisch typisiert, kompiliert
- Nebenläufigkeit
- Garbage Collection





Hallo Welt!

```
1 package main
2
3 import "fmt"
4
5 func main() {
6     fmt.Println("Hallo Welt")
7 }
```

Programm übersetzen / ausführen

```
% go build -o hallo hallo.go
% ./hallo
Hallo Welt

% go run hallo.go
Hallo Welt
```

Pakete / Bibliotheken

```
1  package zeichenkette
2
3  func Rückwärts(s string) string {
4      r := []rune(s)
5      for i := 0; i < len(r)/2; i++ {
6          j := len(r) - i - 1
7          r[j], r[i] = r[i], r[j]
8      }
9      return string(r)
10 }
```

Pakete verwenden

- `$GOPATH/src` ist der Suchpfad für extra Pakete
- `import "<paketname>"`

```
1  package main
2
3  import (
4      "fmt"
5
6      zk "tokkee.org/zeichenkette"
7  )
8
9  func main() {
10     fmt.Println(zk.Rückwärts("Hallo Welt"))
11 }
```

Fehlerbehandlung

```
1  func machEs() (*Object, error) {
2      // ...
3      if err := run(); err != nil {
4          return nil, err
5      }
6      return o, nil
7  }
8
9  func main() {
10     o, err := machEs()
11     if err != nil {
12         log.Exitf("Fehler: %v\n", err)
13     }
14     // ...
```

Die Standard-Bibliothek

`https://golang.org/pkg/`

- Crypto
- Datenbanken
- Go Parser
- Netzwerk, HTTP, SMTP, etc.
- Datenstrukturen

Ein Webserver

```
1  func main() {
2      http.HandleFunc("/hallo", sageHallo)
3      log.Fatal(http.ListenAndServe(":9999", nil))
4  }
5
6  func sageHallo(w http.ResponseWriter,
7      r *http.Request) {
8
9      fmt.Fprintf(w, "Hallo␣%s", r.RemoteAddr)
10 }
```

Nebenläufigkeit

Alle Anfragen werden nebenläufig behandelt.

- Goroutinen
  - `go f(args)`
  - leichtgewichtige Threads
  - hunderte oder tausende werden effizient verwaltet
- Channels
  - Kommunikation zwischen Goroutinen
- `select`
  - warten auf I/O und Channels

Goroutinen

```
1  errCh := make(chan error, 2)
2
3  go func() {
4      errCh <- machEs()
5  }()
6  go func() {
7      errCh <- undDas()
8  }()
9
10 go machEtwasImHintergrund()
```

Goroutinen (2)

```
1  timeout := time.After(50*time.Millisecond)
2  for i := 0; i < cap(errCh); i++ {
3      select {
4      case err := <- errCh:
5          if err != nil {
6              return err
7          }
8      case <-timeout:
9          return fmt.Errorf("timeout(%d)", i)
10     }
11 }
```

Werkzeuge

Go ist dazu gedacht, in Werkzeugen verwendet zu werden (`go/ast`, etc.)

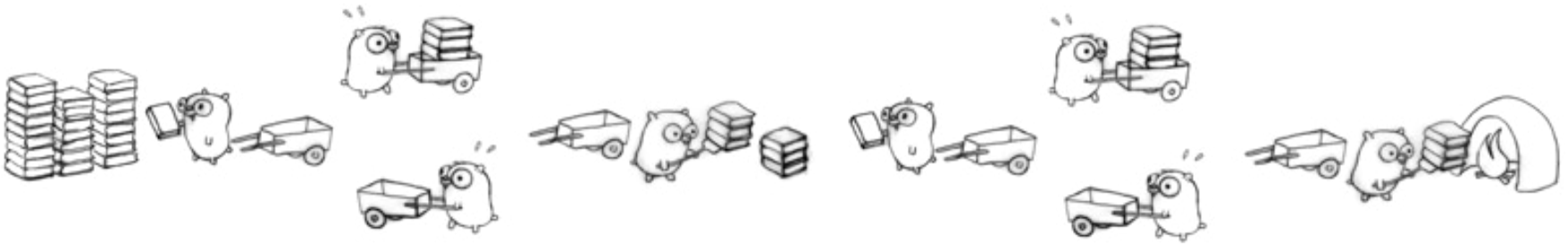
- gofmt, goimports
- godoc, `https://godoc.org/`
- IDE und Editor Unterstützung

vim:

```
1 autocmd filetype go
2     \ autocmd BufWritePre <buffer> Fmt
3 let g:gofmt_command = "goimports"
```

Danke für die Aufmerksamkeit

Fragen, Kommentare?



`https://tour.golang.org` — `https://play.golang.org`